Drive Responsibly Technical Report

Phase 3

Team 2 - 11 am

Musab Abdullah, Sophia Cespedes, Ethan Houston, Matthew Jagen, Kevin Li

15 November 2021

CS 373: Software Engineering

Glenn Downing

The University of Texas at Austin

Motivation

The rise in breweries gives pause to civilians - are they in any way hurting the safety of the city? The website will examine the relationship between Texas counties, the breweries within them, and the frequency of traffic incidents. With an in-depth look at the nature, severity, and frequency of traffic incidents and the number of breweries, both on the county level, users can deduce potential patterns or conclusions from the comparison of the data.

Customer User Stories

The following user stories were provided by our customer group, My NutriPal.

Phase 1

User Story #1: Flags for high numbers of traffic incidents

User Request: I want to be able to easily tell if a given county has too many traffic incidents occurring, maybe by using flags that convey a certain threshold of incidents (>100 incidents, >500 incidents).

Implementation: We are keeping track of the county of each traffic incident using the traffic API, so we can use that data that we've collected to count the number of incidents per county and then add the flags. If we need to, we can also add a timeframe for the incident flags using data from the same API.

User Story #2: Mapping the probability of incidents

User Request: I want to use the accident data from Drive Responsibly with google maps to see what roads have a high likelihood of traffic incidents so I can avoid them and take the safest route.

Implementation: The traffic API provides us with the type of incident (such as congestion or car accident) as well as the location via latitude/longitude and street name. Using this we can map the car accidents using the location data to show which streets are the safest.

User Story #3: Flags for number of breweries in a county

User Request: I would like counties that reach certain milestones for a total number of breweries to be signified with different flags, making them easily distinguishable at a glance.

Implementation: Just as with traffic incidents, one of the attributes we are tracking for

breweries is the county it is in, so we can use this data to count the number of breweries per county and then add the flags.

User Story #4: Graph of traffic incidents over time

User Request: I would like to have a graph that displays the change in incidents over time. This could then be used to see how specific events may have led to fluctuations in the number of accidents.

Implementation: We can already associate traffic incidents with counties using the location data provided by the traffic API. This, along with the time data provided, can be used to create a graph of traffic incidents over time for each county.

User Story #5: Stats page

User Request: I suggest that you make a stats page with significant figures from different years and different demographics to emphasize the severe negative impact of drunk driving.

Implementation: the traffic API doesn't track any data relating to the people that may have been a part of the traffic incidents so we can't do much about demographics other than generalizations about the county it was in. We could still make a stats page with different data relating to the location of the incidents relative to breweries such as the average number or severity of incidents within a given range of a brewery.

Phase 2

User Story #1: Add filter for different brewery types

User Request: Filter by different craft beers, breweries, etc. Etc. lagers, IPAs, pale ale, porters...

Implementation: Filtering is out of scope for phase 2, but we will certainly look into implementing this once we implement filtering and sorting in phase 3.

User Story #2: Add traffic density map for historical accidents

User Request: Display a map that shows the frequency of incidents that happened on roads near breweries to indicate the importance of responsible driving near breweries.

Implementation: Displaying a map that shows the frequency of traffic incidents as opposed to just the location of nearby instances of traffic incidents was out of the scope of phase 2. Instead, we will incorporate this by showing the location of specific traffic incident instances on the map as opposed to frequency.

User Story #3: Distribution of time of day for traffic incidents

User Request: Allow users to visualize what time of day the most frequent traffic incidents are.

Implementation: Creating a visualization of traffic incidents based on the time of day was out of scope for phase 2 of this project, so instead we will allow users to filter traffic incidents by time in phase 3. We believe this implementation will also better represent our motive in creating DriveResponsibly as this will allow users to additionally filter and sort by incident severity and involvement of a drunk driver.

User Story #4: Create a tips page on how to avoid incidents based on the most frequent times

User Request: Based on the data that was gathered, determine whether or not a user should drive during that time period. This can help reduce accidents and make driving safer. Overall, just based on the client's timezone, suggest whether they should drive or not.

Implementation: The creation of a static tips page is not as involved to the user as being able to interact with the actual data itself. So, to better represent the relationship between traffic incidents and breweries, we will implement this by allowing users to filter and sort by the different attributes of traffic incidents in phase 3.

User Story #5: Filter based on demographic of accident

User Request: Create a filter for the demographics involved in the accident. Particularly, the gender, age, and ethnicity of people who caused/were injured within the accident. This can allow for further data analysis in the future.

Implementation: The data we have for traffic incidents does not track the identity of those involved, so unfortunately we are unable to implement this. However, you will be able to filter by involvement by a drunk driver once filtering and sorting are implemented in phase 3.

User Story #6: Create analysis for vehicles involved in accidents

User Request: This can be represented by a graph. For example, the car make, the color, the model, the year it was built. Provide a visual to determine what type of cars are more likely to get into an accident.

Implementation: The traffic incident API that we scraped for our traffic incident information does not directly track any data on the cars involved in the incident, so we are unable to show this data on our site at the moment. If we find this data later on and can tie it to the incident id, we will be sure to add this in a later phase.

Phase 3

User Story #1: Fancy Search Bar

User Request: A lot of websites I've used have created a better experience through responsive search bars. For the phase in which you manage how the user interacts with the search bar, add effects to make the experience more responsive. For example, you can make the search bar glow, shrink, or animate when typing. The assessment for this is that it should take around 1 hour to implement.

Implementation: Unfortunately this was unable to be added to this phase despite being very doable because we received this user story very close to the phase 3 deadline at 7PM Tuesday. We will certainly revisit it in the next phase. Our estimated time to complete is one hour.

User Story #2: Create live map for accidents

User Request: As a whole, I appreciate the relationship you all are developing between crash accidents and breweries. To take your website to the next level, I think it would be really useful to see crash accidents as they happen in real time. Perhaps if there is some API, you could create a map that displays this information in real time. I estimate this to take 4-6 hours.

Implementation: Our database and API are not able to support real time information on accidents, so we can't implement a live map on our site. However, the incident page instances each have a map of where the incident occurred and the time period when traffic was obstructed.

User Story #3: Fix margins on home page

User Request: The home page displays text and three images underneath. While the concept is a good one for the layout, everything feels a bit cramped and is hard to digest as a user. Can you fix the padding and margins for the home page so it will flow better and look more visually appealing? This should take around 1 hour.

Implementation: We definitely agree that our home page can use a redesign but since this user story was received so close to the phase 3 deadline, we are unable to fit the redesign into phase 3. In the next phase we will revisit this to make better use of all the whitespace and make the landing page more user friendly. Our estimated time to complete is one to two hours.

User Story #4: Fix incidents table styling

User Request: I love the design for most of your website and the material theme. However, the incidents page has a black table that's starkly different from the rest of the UI. Can you redesign the table to follow the lighter theme of rounded corners? This should take 30 minutes to 1 hour.

Implementation: We agree that the stark contrast between the county and brewery page design and the incident page design needs to be addressed. Unfortunately, the changes didn't make it in for phase 3 because of the little time we had between receiving the user

story and phase 3 completion. We will come back next phase to update the incident table styling. Our estimated time to complete is one hour.

User Story #5: Center bottom two elements on about page

User Request: In general, the styling for the about page is decent. However, the bottom two elements with Sophia and Matthew have a bit of a weird padding. Can you fix this by either centering them or making them left aligned? This should take around 30 minutes.

Implementation: We updated the spacing of the developer team cards on the about page so that they are better-aligned as per the user story. Additionally, the cards now behave in a more user-friendly way when shrinking or growing the browser window. Our estimated time to complete was 30 minutes. The actual time to complete was 20 minutes.

Developer User Stories

The following user stories were provided to our developer group, We Like Sportz.

Phase 1

User Story #1: Featured page for well-known teams/players

> **User Request**: As someone who isn't very familiar with sports teams in general, I want to easily be able to see what teams and players are good or well known. Can you add a featured page that shows players that are widely considered great or maybe teams that have won some sort of championship before? I think this will help to broaden the reach of your website to casual fans.
>
> **Developer Response**: We were planning to implement similar functionality, but instead of creating a featured page, we intend to make the teams' instances sortable by the number of championships they have won in Phase 3 so that users can view the most successful teams for each sport. As for players, we include a list of the notable players for each sports team on the instance page as well as links to the players pages. Does that sound good to you?

User Story #2: Filtering pages for a specific sport

> **User Request**: As someone who only likes a specific sport, I want to be able to only see cities that have an official team for that sport. Can you add a filter to your cities page that allows me to filter out all of the cities that don't have a team for the sport I have checked? I think this will make your site much easier to use for me and other people who only care to look up information for the specific sport they like.
>
> **Developer Response**: Filtering the cities by sports they have is a feature we will look to incorporate in Phase 3 when filtering and sorting are required. We currently have begun working on filtering and sorting and have a prototype, but it isn't functional at the moment. We will complete it if we have time towards the end of Phase 2.

User Story #3: Filtering players by retired/active

User Request: As someone who likes knowing the history of sports players, I would like to see a list of players that are retired and hide active players. Can you add a filter to your players' page that allows me to see only retired players? I think this will also be useful for people that only care about active players and want to filter out retired players as well.

Developer Response: Filtering the players by whether they are retired or active is a feature we will definitely look to incorporate in Phase 3 when filtering and sorting are required. However, we may not be able to find this information and include it in our database, in which case we will only display active players on our website.

User Story #4: Player comparison page

User Request: As an avid power user of WeLikeSportz, I often have multiple tabs pulled up side by side to compare players. Could you add a page that can compare the stats of two players side by side for me? I think having this head-by-head built into your site will make comparing the stats of two players much easier and enjoyable.

Developer Response: Since we're currently only displaying static information for Phase 1 we won't be able to implement this feature since it is out of scope. Rather than creating a separate page to compare player stats side by side, would it be ok if instead, we make the players' instances sortable/filterable instead, so that way you can see which players have the highest W/L record, championships, etc.? However, this feature probably won't be implemented until at least Phase 3 since it involves filtering and sorting.

User Story #5: Recent game scores and win/loss

User Request: As someone who keeps up with sports very closely, I want to be able to see recent scores for players/teams. Can you add a list of scores and wins/losses on each player or team page that is sorted by most recent? This will make your site a go-to for my day-to-day following of sports content.

Developer Response: I agree that having recent games scores and win/loss would be a cool feature to have. Unfortunately, given our data sources and the APIs publicly

available to us, we weren't able to find this information and include it in our database, so we won't be able to implement this feature, at least for Phase 1/2.

Phase 2

User Story #1: Show where you can watch a team's games on the team page

User Request: As someone who likes to watch sports in my home, I wish there was an easy way to see where I can watch certain teams. Could you change the team's page to show what broadcasting service airs their games? This would make it very easy for me to find out where I can watch my favorite teams.

Developer Response: Hey Matthew, I agree that having a link to where you can watch a team's games would be a cool feature to have. Unfortunately, given our data sources and the APIs publicly available to us, we weren't able to find this information and include it in our database, so we won't be able to implement this feature, at least for Phase 2. However, we do have plenty of other sources of media for teams such as a link to their Facebook page, where you can potentially find more info about their matches.

User Story #2: Show games that happened in a city on the city page

User Request: As someone who can't travel out of town to watch sports games, I'm not interested in any games outside of my city. Could you show a list of matches that happened or will happen in a certain city on that city's page? It would become so much easier to see what games I can go to.

Developer Response: Hey Matthew, I agree that having games that happened in a city would be a cool feature to have. Unfortunately, given our data sources and the APIs publicly available to us, we weren't able to find this information and include it in our database, so we won't be able to implement this feature, at least for Phase 2. We will try to expand our database but cannot guarantee that this feature will be added.

User Story #3: Show a player’s team history on their player page

User Request: As someone who is a big fan of a few players, I'd like to know more about my favorite players' past. Could the player page include a list of previous teams they were on as well? It's interesting to see where the players I like so much came from and who they played for before.

Developer Response: Hey Matthew, I agree that having a player's team history would be a cool feature to have. Unfortunately, given our data sources and the APIs publicly available to us, we weren't able to find this information and include it in our database, so we won't be able to implement this feature, at least for Phase 2.

User Story #4: Show team’s home stadium on the team page

User Request: As someone who likes to watch sports in person with friends, I would like to use your website to help plan when we can go watch. Could you include the address of the team's home stadium on the team page? This would help us plan ahead of time and possibly use it to look up food places close by since stadium food is so expensive.

Developer Response: Hi Matthew, thanks for the feedback! We were able to add the name of the stadium for each Team in the team instance pages. Enjoy!

User Story #5: Sort or filter players by jersey number

User Request: As a young athlete, I have to pick a number for my team jersey soon. Could I sort or filter players by jersey number? It would be really cool if I could see what great players used each number and would help me pick mine.

Developer Response: Hi Matthew, filtering the players by jersey number is a feature we will definitely look to incorporate in Phase 3 when filtering and sorting are required. We currently have begun working on filtering and sorting and have a prototype, but it isn't functional at the moment. We will complete it if we have time towards the end of Phase 2.

Phase 3

User Story #1: Filtering by multiple cities

User Request: I travel frequently for my work, and so I have many cities that I would like to see sports info for. When filtering by cities, could you make it so that I can 'check' which cities I would like to see on the browse teams page? This would make it much easier to see only the teams from cities I like to follow.

User Story #2: Showing team home stadium on map

User Request: One of my favorite hobbies is going to all kinds of sports games. I see that you have the team's home stadium on their page, but I think it would be very helpful if the page could show the location of the stadium on a map (if applicable). Just like the city page, I think the map visualization would add a lot to the site.

User Story #3: Miscellaneous sorting options

User Request: I really enjoy the more niche 'fun-facts' about sports teams and players. Could you add more filter options so we can easily sort for info like tallest player, heaviest player, or oldest player? I think the additional options to sort by age, height, weight, etc., would make your site more interesting for people that like fun sports trivia.

User Story #4: Link to recent matches or league standings

User Request: I enjoy using WeLikeSportz for info on players and teams, but I feel like finding recent matches or league standings is fairly difficult. Would it be possible to link the team's "League" field to a page with the current standings (or most recent standings) for that league? Including the info on the site would probably clutter the pages too much, but linking to where the user can find the info would be great!

User Story #5: Searching for city

User Request: I live in a city that is currently buried into the later pages of the cities table on the cities page. Could you add a search bar for cities on the page with the city

table? This would make it take much less time for me to navigate to the cities I want to see on your website.

RESTful API

Postman Documentation: https://documenter.getpostman.com/view/10582451/UUy4ckUf

Model Endpoints

The following endpoints will return all of the instances in the specified model.

Breweries

- GET https://www.driveresponsibly.me/api/breweries?page_size&brewery_type&city&min_created_at&max_created_at&state&min_latitude&max_latitude&min_longitude&max_longitude
    - Returns all of the breweries in Texas. There are a list of query parameters that the user can include to specify what breweries to show, all of which are documented on the provided Postman documentation:
        - page_size
        - brewery_type
        - city
        - min_created_at
        - max_created_at
        - state
        - min_latitude
        - max_latitude
        - min_longitude
        - Max_longitude

APIs Used

- https://api.openbrewerydb.org/breweries
    - Used to scrape most of the attributes for breweries
    - We used this endpoint to get all the breweries in Texas

    - Latitude/longitude fields allowed us to connect each brewery with the closest traffic incidents
- Zip code database csv
    - This was used to help map each zip code of a brewery to a county
    - We used this to add a county field on each brewery that would act as a foreign key to a county to help us connect the two models.

Counties

- GET https://www.driveresponsibly.me/api/counties?page_size&min_P1_001N&max_P1_001N&GEO_ID&state&min_H1_002N&max_H1_002N&min_H1_003N&max_H1_003N
    - Returns all of the counties in Texas. There are a list of query parameters that the user can include to specify what counties to show, all of which are documented on the provided Postman documentation:
        - page_size
        - min_P1_001N
        - max_P1_001N
        - GEO_ID
        - state
        - min_H1_002N
        - max_H1_002N
        - min_H1_003N
        - max_H1_003N

APIs Used

- https://api.census.gov/data/2020/dec/pl
    - Used to scrape basic data about various counties
    - Returned data in json format where the key was an unreadable encoding of a type of attribute and the value was data values. We mapped each encoding to its corresponding human readable attribute in our scraping process

Traffic Incidents

- GET https://www.driveresponsibly.me/api/traffic_incidents?page_size&type&severity&eventCode&min_distance&max_distance&min_delayFromFreeFlow&max_delayFromFreeFlow
    - Returns all of the traffic incidents in Texas. There are a list of query parameters that the user can include to specify what traffic incidents to show, all of which are documented on the provided Postman documentation:
        - page_size
        - type
        - severity
        - eventCode
        - min_distance
        - max_distance
        - min_delayFromFreeFlow
        - max_delayFromFreeFlow

APIs Used

- https://developer.mapquest.com/documentation/traffic-api/incidents/get/
    - Used to scrape all our data regarding traffic incidents
    - Latitude/longitude helped us connect each traffic incident to nearby breweries
    - Location data also helped us connect each incident to its respective county

Instance Endpoints

Brewery

- GET https://www.driveresponsibly.me/api/breweries/:Id
    - Returns the brewery with the provided id (required).

County

- GET https://www.driveresponsibly.me/api/county/:Id
    - Returns the county with the provided id (required).

Traffic Incident

- GET https://www.driveresponsibly.me/api/traffic_incident/:Id
    - Returns the traffic incidents with the provided id (required).

Models

Breweries - 343 in Texas[1]

- ID - the unique brewery ID. (int)
- Name - the name of the brewery. (string)
- Type - the type of brewery (micro, nano, regional, brewpub, large, planning, bar, contract, proprietor, closed)[2].
- Street - the street address of the brewery. (string)
- Postal Code - the postal (or zip) code of the brewery. (string)
- County - the county that the brewery is located in. (string)
- City - the city that the brewery is located in. (string)
- State - the state the brewery is located in. (string)
- Latitude - the latitudinal position of the brewery in degrees. (floating point number)
- Longitude - the longitudinal position of the brewery in degrees. (floating point number)
- Phone Number - the brewery's phone number. (10 digit integer)
- Website URL - the brewery's website URL. (string)

Counties - 254 in Texas

- ID - the unique county ID. (int)
- Name - the name of the county (string)
- Median Household Income - the median of all of the county's household incomes (int)
- Average High School Educational Attainment - the average percent of citizens in the county with a high school diploma (floating point number)
- Average Baccalaureate Educational Attainment - the average percent of citizens in the county with a bachelor's degree (floating point number)
- Unemployment Levels - the unemployment rate of the county (floating point number)
- Poverty - the number of citizens in the county living in poverty (int)
- Population - the number of people living in the county. (int)

---

[1] Brewery DB GitHub: https://github.com/openbrewerydb/openbrewerydb/tree/master/data/united-states
[2] List of Breweries: https://www.openbrewerydb.org/documentation/01-listbreweries#by_type

Traffic Incidents - about 1,000

- ID - the unique incident ID. (int)
- county_id - the county the incident occurred in. (county_id)
- City - the city that the incident occurred in. (string)
- Drunk Driver - whether or not the incident involved a drunk driver (boolean)
- Fatalities - the number of deaths from the accident. (int)
- Number of People - the number of people involved in the accident. (int)
- Road Type - the type of road where the incident occured. (int)
- Time Period - the date, start and end time of the traffic incident. (DateTime)
- Latitude - the latitudinal position of the incident in degrees. (floating point number)
- Light Condition - the level of brightness at the time of the accident (string)
- Longitude - the longitudinal position of the incident in degrees. (floating point number)
- State - the state that the incident occurred in. (string)

Filterable Attributes

Breweries

- Type
- Website
- Phone number

Counties

- Population
- Poverty

Traffic Incidents

- Light Conditions
- Drunk Driver

- Area

Sortable Attributes

Breweries

- City
- Name

Counties

- Name
- Population
- Poverty
- Unemployment

Traffic Incidents

- City
- Fatalities

Media

Breweries

- Related Tweets
- Map

Counties

- Related Tweets
- Map

Traffic Incidents

- Related Tweets
- Map

Connections

Breweries

- Each brewery is located within a county. The county id acts as a foreign key. This is a many-to-one relationship.
- Each brewery is connected to traffic incidents via an association table based around the idea of K nearest neighbors. We store the closest traffic incidents to each brewery and the associated distance. This is a many-to-many relationship.

Counties

- Counties are connected to both breweries and traffic incidents as both are located within a particular county.
- We have backrefs from a county to traffic incidents and breweries. This is a one-to-many-relationship.

Traffic Incidents

- Traffic incidents are pretty similar to breweries in terms of their relationship schema.
- Each traffic incident is located within a county. The county id acts as a foreign key. This is a many-to-one relationship.
- Each incident is connected to breweries via an association table based around the idea of K nearest neighbors. We store the closest breweries to each incident and the associated distance. This is a many-to-many relationship.

Phase Features

Phase 2

Database

We implemented the database by using the AWS Relational Database Service[3] (Amazon RDS)[4] and chose MySQL to be our database language. We wrote Python class definitions for each model (Brewery, County, TrafficIncident, and BreweryTrafficIncident), all of which we declared as subclasses of db.model where db is an instance of the SQLAlchemy() class from flask_sqlalchemy.[5] For Brewery and TrafficIncident, we included a db.relationship call to connect the Brewery and TrafficIncident models, making it easier for BreweryTrafficIncident to find the correlation between breweries and traffic incidents (if any). After we had our db session set up (adding the tables, calculating the distances between breweries and traffic incidents, etc.), we were able to create our database using sqlite3. We then went ahead and created an Amazon RDS instance. However, sending the .sql file as it was to the Amazon RDS did not work; after researching the issue online, we were able to find a script that could convert our .sql file that was compatible with sqlite3 to a .sql file that was compatible with MySQL.[6] Once we were able to send our database to the Amazon RDS, we were able to modify the database on our end and send the results back to the RDS. Additionally, we were able to configure our Flask app to use our Amazon RDS URL for the "SQLALCHEMY_DB_URI" instead of the one provided by sqlite3 (our initial URL before we made the RDS).[7]

Pagination

We implemented the pagination for our instance pages by utilizing the flask_restless.APIManager class from Flask-Restless-NG. With this class, we were able to create

---

[3] Burnin' Up Technical Report: https://gitlab.com/caitlinlien/cs373-sustainability/-/blob/master/Technical%20Report.pdf
[4] Amazon RDS User Guide: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html
[5] Flask-SQLAlchemy: https://flask-sqlalchemy.palletsprojects.com/en/2.x/api/?highlight=sqlalchemy
[6] Sqlite3 to MySQL: https://stackoverflow.com/questions/18671/quick-easy-way-to-migrate-sqlite3-to-mysql
[7] Burnin' Up Technical Report

an API blueprint[8]; when initializing the blueprint, we were able to set one of the optional parameters called “page_size”[9] to the number of instances of each model that we wanted to show on any given page. In our case, we chose twelve to be our pagination (i.e., there are twelve instances of the Brewery model on page 1 of the Brewery main page, another twelve instances on page 2, etc.).

Phase 3

Sorting

Sorting was implemented by Flask-Restless-NG. We defined the resources (county, incident, brewery) for sorting. In our API calls to the backend, we added query parameters to tell Flask-Restless-NG how to sort the data.

Searching

Searching was implemented client-side with basic usage of the built-in javascript string method `includes()`. This method was not our first choice, however the existing design of both our React app and the backend handlers made it difficult to implement search due to prior design decisions, which is a learning point.

Filtering

Filtering was implemented by Flask-Restless-NG. We defined the resources (county, incident, brewery) for filtering. In our API calls to the backend, we added query parameters to tell Flask-Restless-NG how to filter the data.

---

[8] Flask-Restless: https://flask-restless.readthedocs.io/en/stable/api.html?highlight=blueprint#module-flask.ext.restless

[9] Flask-Restless-NG: https://flask-restless-ng.readthedocs.io/en/latest/api.html?highlight=create_blueprint#flask_restless.APIManager.create_api_blueprint

Tools

Data

Open Brewery DB API: Used to get information on Texas breweries.

- Data Scraping: Data was downloaded from the *openbrewerydb* website.

USDA County-Level Data Sets: Used to get information on Texas counties.

- Data Scraping: Multiple .csv files were downloaded for each of the different data endpoints (poverty, population, unemployment, average high school educational attainment, average bachelor's degree educational attainment) from the *USDA* website.

US Decennial Census API: Used to get the list of Texas counties.

- Data Scraping: Data was downloaded from the US Census website; afterwards, we recorded the names of the Texas counties from the data.

Google Geocoding API: Used to find location from a given latitude and longitude.

- Data Scraping: .

Mapquest Traffic API: Used to get information on traffic incidents in Texas.

- Data Scraping:

Backend

Docker: Used to package the backend into a container.

Flask: Used for our backend framework.

Flask-Restless-NG: Used the routes provided (plus additional custom routes that we created) and the APIManager class to create an API for our database.

Postman: Used to document our API.

Frontend

AWS: Used AWS to host the frontend; this included RDS, EC2, and Elastic Beanstalk.

Docker: Used to package the frontend into a container.

React: Used for our frontend development framework.

Others

Discord: Used for group communication.

Ed Discussion: Used for class communication.

GitLab: Used to store and manage the project repository, issue tracking, and code collaboration.

Grammarly: Used to proof-read the technical reports.[10]

[10] Made the changes recommended by Grammarly on the document when possible (this included editing the Customer User Stories we were provided).

Hosting

The site is hosted at: https://www.driveresponsibly.me/#/

The website is hosted using AWS, and our domain was acquired from NameCheap.

Sources[11]

AWS RDS

AWS Database Migration Service:

> https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html

Connect to RDS Instance:

> https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html

Local to RDS DB Transfer:

> https://stackoverflow.com/questions/11731714/how-do-i-import-a-local-mysql-db-to-rds-db-instance

MySQL DB Setup:

> https://aws.amazon.com/getting-started/hands-on/create-mysql-db/

Store Data in MySQL:

> https://medium.com/@eric_landstein/using-python-to-store-data-in-mysql-on-aws-rds-77a524a0804d

[11] Some of the links are to the main page of a given resource, but sections or hyperlinks from the main page were used as well.

Data

Brewery API:

> https://www.openbrewerydb.org/documentation/01-listbreweries#by_state

County .csv File:

> https://www.ers.usda.gov/data-products/county-level-data-sets/download-data/
>
> Used wording on the datasets and other data tools provided by the resource to write the descriptions for the *County* model in the *Models* section.

County API:

> https://www.census.gov/data/developers/data-sets/decennial-census.html

Latitude/Longitude API:

> https://developers.google.com/maps/documentation/geocoding/overview#ReverseGeocoding

Traffic Incident API:

> https://developer.mapquest.com/documentation/traffic-api/incidents/get/

DB Diagram

Documentation:

> https://www.notion.so/dbdiagram-io-Docs-Public-d405d7d9efd4419d9b6402eb6feb8cd1
>
> https://www.dbml.org/docs/

Many-to-Many Relationship:

> https://community.dbdiagram.io/t/tutorial-many-to-many-relationship/412

Online Workspace:

https://dbdiagram.io/d

Docker

Compose Environment Variables:

https://docs.docker.com/compose/environment-variables/

Create Env Variables:

https://docs.docker.com/compose/env-file/

How to run SQL server in a Docker container:

https://www.youtube.com/watch?v=RAE-VcZ3u2A

Microsoft SQL Server:

https://hub.docker.com/_/microsoft-mssql-server

Use Docker image to create SQL server:

https://docs.microsoft.com/en-us/sql/linux/quickstart-install-connect-docker?view=sql-server-ver15&pivots=cs1-bash

Flask

AWS Guide by Jefferson Ye on GitHub:

https://github.com/forbesye/cs373/blob/main/Flask_AWS_Deploy.md

Documentation:

https://flask.palletsprojects.com/en/2.0.x/

Flask-Restless

Documentation:

https://flask-restless.readthedocs.io/en/stable/

https://flask-restless.readthedocs.io/en/latest/api.html

Fetching:

https://flask-restless-ng.readthedocs.io/en/latest/fetching.html#fetching-resources-and-relationships

Flask-Restless-NG Documentation:

https://flask-restless-ng.readthedocs.io/en/latest/

https://github.com/mrevutskyi/flask-restless-ng

https://pypi.org/project/Flask-Restless-NG/

Pagination:

https://flask-restless.readthedocs.io/en/stable/customizing.html#server-side-pagination

Search:

https://flask-restless.readthedocs.io/en/stable/searchformat.html#searchformat

Flask SQLAlchemy

Many-to-Many Relationship Example 1:

https://flask-sqlalchemy.palletsprojects.com/en/2.x/models/

Many-to-Many Relationship Example 2:

https://gist.github.com/kphretiq/80a1babea45cdf952008

Set-Up:

https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/

Queries:

https://flask-sqlalchemy.palletsprojects.com/en/2.x/queries/

HTML

<dl>:

https://html.spec.whatwg.org/multipage/grouping-content.html#the-dl-element

Documentation:

https://developer.mozilla.org/en-US/docs/Web/HTML

Flow Content:

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Content_categories#flow_content

Tutorial:

https://html.com/

Unknown Unicode Error:

https://en.wikipedia.org/wiki/Percent-encoding

https://www.compart.com/en/unicode/U+FFFD

JavaScript

Convert to String:

https://medium.com/dailyjs/5-ways-to-convert-a-value-to-string-in-javascript-6b334b2fc778

Decimal Truncation:

https://www.jsdiaries.com/how-to-remove-decimal-places-in-javascript/

Guide:

https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript Basics

> https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

Learning Resource:

> https://www.javascript.com/learn/strings

Lists and Keys:

> https://reactjs.org/docs/lists-and-keys.html

Locale String:

> https://stackoverflow.com/questions/2901102/how-to-print-a-number-with-commas-as-thousands-separators-in-javascript/17663871#17663871

Map:

> https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Jest

Documentation:

> https://jestjs.io/docs/tutorial-react

JSON

Filtering and Sorting:

> https://jsonapi.org/format/#fetching-sorting

MySQL

APT Repo:

https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/

Different MySQLClients:

https://stackoverflow.com/questions/43102442/whats-the-difference-between-mysqldb-mysqlclient-and-mysql-connector-python

Dump Data into SQL File:

https://dbconvert.com/sqlite/mysql/

Fix to *ERROR: Could not find a version that satisfies the requirement MySQLdb (from versions: none)*:

https://stackoverflow.com/questions/34030215/error-loading-mysqldb-module-and-pip-install-mysqldb

Fix to *ModuleNotFoundError: No module named 'MySQLdb'*:

https://stackoverflow.com/questions/454854/no-module-named-mysqldb

MySQLClient:

https://pypi.org/project/mysqlclient/

MySQL-python:

https://pypi.org/project/MySQL-python/#description

Start Guide:

https://dev.mysql.com/doc/mysql-getting-started/en/

Sqlite Dump:

https://www.sqlitetutorial.net/sqlite-dump/

Sqlite3 to MySQL:

https://stackoverflow.com/questions/18671/quick-easy-way-to-migrate-sqlite3-to-mysql

npm

Documentation:

https://docs.npmjs.com/cli/v7/commands/npm-start

Pip

Documentation:

https://pip.pypa.io/en/stable/

Installation:

https://packaging.python.org/tutorials/installing-packages/#ensure-you-can-run-pip-from-the-command-line

PlantUML

Class Diagram Documentation:

https://plantuml.com/class-diagram

Colors:

https://plantuml.com/color

Common Commands:

https://plantuml.com/commons

Component Diagram Documentation:

https://plantuml.com/component-diagram

Example:

https://blog.jetbrains.com/dotnet/2020/10/06/create-uml-diagrams-using-plantuml/

How to align blocks:

https://stackoverflow.com/questions/11557426/how-to-align-blocks-in-plantuml-class-diagrams

How to get class diagrams working on VS Code:

https://stackoverflow.com/questions/53856294/plantuml-extension-for-visual-studio-code-on-windows-only-working-with-sequence

PDF Guide:

http://plantuml.com/guide

Preprocessing:

https://plantuml.com/preprocessing

skin params:

https://plantuml-documentation.readthedocs.io/en/latest/formatting/all-skin-params.html#d

Use with VS Code Extension:

https://towardsdatascience.com/drawing-a-uml-diagram-in-the-vs-code-53c2e67deffe

Postman

Documentation Example 1:

https://documenter.getpostman.com/view/12084061/Tz5jdKfE#c85fd3a3-8a18-403e-9b22-7f6c9a560ade

Documentation Example 2:

https://documenter.getpostman.com/view/12123261/TVRdAWse

Documentation Example 3:

https://documenter.getpostman.com/view/12817007/TVYPztiN

Ed Discussion Post #184

Ed Discussion Post #191

Test Scripts:

https://learning.postman.com/docs/writing-scripts/test-scripts/

React

Components and Props:

https://reactjs.org/docs/components-and-props.html

DOM Elements:

https://reactjs.org/docs/dom-elements.html#style

Effect Hook:

https://reactjs.org/docs/hooks-effect.html

How to create a React app:

https://www.codecademy.com/articles/how-to-create-a-react-app

Pagination:

https://www.digitalocean.com/community/tutorials/how-to-build-custom-pagination-with-react

https://github.com/AdeleD/react-paginate

https://www.npmjs.com/package/react-paginate

react-google-maps:

https://www.npmjs.com/package/react-google-maps

React.Component:

https://reactjs.org/docs/react-component.html

React and Javascript:

https://reactjs.org/docs/getting-started.html

Render:

https://reactjs.org/docs/rendering-elements.html

State Hook:

https://reactjs.org/docs/hooks-state.html

Testing Library:

https://testing-library.com/docs/react-testing-library/intro/

Travel Scares Me - Spring 2021, 11 a.m. Group (CS373): dropdown buttons and layout

https://www.travelscares.me/countries

Note: The design choices for the dropdown buttons gave us an idea for our dropdown button design.

React-Bootstrap

Cards:

https://react-bootstrap.github.io/components/cards/

https://getbootstrap.com/docs/4.0/components/card/

Dropdowns:

https://react-bootstrap.github.io/components/dropdowns/

Split Buttons:

https://react-bootstrap.netlify.app/components/dropdowns/#split-button-dropdowns

Learning Guide/Documentation:

https://getbootstrap.com/docs/5.0/content/typography/

https://react-bootstrap.netlify.app/getting-started/introduction/

https://react-bootstrap.github.io/

Pagination:

https://react-bootstrap-table.github.io/react-bootstrap-table2/docs/pagination-props.html

https://thewebdev.info/2020/08/01/react-bootstrap%E2%80%8A-%E2%80%8Apagination-and-progress-bar/

RESTful API

Definition:

https://www.ibm.com/cloud/learn/rest-apis

Filtering/Searching/Sorting:

Ed Discussion #263:

> For sorting/filtering/searching, we want you to implement it through the backend, which means that you need to implement it using API query parameters. Your frontend would then use your API + query parameters, make that request, and then present the query returned. You would be implementing these queries in your app.py/main.py. Here's an article explaining what API query parameters are: https://rapidapi.com/blog/api-glossary/parameters/query/. Here's an example of an API route with query parameters: api.website.me/model1?attribute1=john%20smith&attribute2=jane%20doe

Query Parameters:

https://rapidapi.com/blog/api-glossary/parameters/query/

Python

.env Files:

https://dev.to/jakewitcher/using-env-files-for-environment-variables-in-python-applications-55a1

Decimal Truncation:

https://stackoverflow.com/questions/29246455/python-setting-decimal-place-range-without-rounding

dotenv:

https://pypi.org/project/python-dotenv/

https://github.com/theskumar/python-dotenv#getting-started

Math Documentation:

https://docs.python.org/3/library/math.html

Selenium

Documentation:

https://www.selenium.dev/documentation/webdriver/

SQLAlchemy

Documentation:

https://www.sqlalchemy.org/

Python:

https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/

Serialize to JSON:

https://stackoverflow.com/questions/5022066/how-to-serialize-sqlalchemy-result-to-json

Technical Report

Books4You - Spring 2021, 11 a.m. Group (CS373):

https://gitlab.com/cs373-group14/books4u/-/blob/master/Technical_Report.pdf

Note: The Books4You technical report was used for formatting this report and deciding what to include/not include. Additionally, the phrase, "filter/sort" used in the Books4You technical report is loosely used in the RESTful API portion of this technical report, particularly the description of the endpoints, including specifying required id. Also, the "Hosting" section of the Books4You technical report was used as a reference for this technical report.

Burnin' Up (CS373):

https://gitlab.com/caitlinlien/cs373-sustainability/-/blob/master/Technical%20Report.pdf

Note: The Burnin' Up technical report was used for formatting this report and deciding what to include/not include. Multiple sections of the Burnin' Up report were used as a reference (e.g., tools section - layout, the position of 'Postman,' the use of 'our' instead of 'the' when describing our use of the data sources, etc.).

Footnotes:

https://www.loyola.edu/academics/history/opportunities/style-guide/~/link.aspx?_id=37C070858EEF41859AB4C9FBC6B44405&_z=z

Format:

https://medium.com/technical-writing-is-easy/how-to-write-technical-report-e935210002c9

Online Example:

https://web.mit.edu/course/21/21.guide/rep-resc.htm

Title Page:

https://web.mit.edu/course/21/21.guide/title.htm

TypeScript

Any Type:

https://ultimatecourses.com/blog/typescript-types-the-any-type

Async and Await:

https://blog.logrocket.com/async-await-in-typescript/

Main Website:

https://www.typescriptlang.org/

Tutorial:

https://www.koderhq.com/tutorial/typescript/

UML Diagram

Around Austin - Spring 2021, 11 a.m. Group (CS373):

https://gitlab.com/jyotiluu/cs373-aroundatx/-/blob/master/uml.png

Aid All Around - Spring 2021, 11 a.m. Group (CS373):

https://gitlab.com/cs373-group23/aid-all-around/-/blob/master/UMLDiagram.png

unittest

Documentation:

https://docs.python.org/3/library/unittest.html#unittest.TestCase